METHODS AND SYSTEMS FOR MANAGING SUCCESSFUL COMPLETION OF A NETWORK OF PROCESSES

Cross Reference to Related Applications

This application claims the benefit of United States Provisional Application Serial No. 60/531,741, filed by Simon Bowie-Briton on December 22, 2003 and entitled "Methods and Systems For Managing Successful Completion of a Network of Processes", which is incorporated herein by reference.

Field of the Invention

The present invention relates generally to data processing systems, and, more particularly, to methods and systems for managing successful completion of a network of processes.

Background of the Invention

A well known difficulty in data processing involves the occurrence of a system failure while a transaction is being processed. For example, a transaction may involve the steps necessary for transferring $100 from a customer's savings account to the customer's checking account. Suppose the $100 was deducted from the customer's savings account, then a system failure occurred before the amount was added to the customer's checking account. The customer's account information would be in error.

There are various conventional techniques to deal with this problem. A very common solution is to employ what is known as a two-phase commit. In such a protocol,

the transaction is designated by a "Begin Transaction" operation. The transaction ends with either a "Commit" operation or a "Rollback" operation. The Commit is used to signal a successful completion of the transaction. The Rollback is used to signal that the transaction was unsuccessful. When a Commit is received, usually a database management system will then write the results to persistent storage (e.g., to DASD).

Although the two-phase commit is often useful, there are certain drawbacks. One major disadvantage is the overhead involved. Furthermore, this approach is not suitable in every processing environment. For instance, when executing a network of processes in which a graph of processing nodes can be dynamically changed, and where no *a priori* knowledge of the graph structure exists, it can be difficult using such an approach to even know when successful completion of the network has occurred.

Summary of the Invention

The present invention involves methods and systems for managing successful completion of a network of processes. The network of processes can be represented as a graph. In this representation, the nodes of the graph represent the processes, and the edges of the graph represent events associated with the processes. Processing starts at the root node, and is based on the result of an initially unknown graph. When an event is to be produced or consumed, a message to that effect is transmitted to a component called a Q-Manager. Using the messages, which are received in event order, the Q-Manager keeps track of the state of the graph, and determines when successful completion of the processing has occurred. Once this occurs, the Q-Manager sends a notification indicating completion of the network.

These and other aspects, features and advantages of the present invention will become apparent from the following detailed description of preferred embodiments, which is to be read in connection with the accompanying drawings.

Brief Description of the Drawings

FIG. 1 illustrates an exemplary Java network architecture;

FIG. 2 illustrates an exemplary Q-Manager for managing successful completion of a network of processes; and

FIG. 3 illustrates changes over time to an exemplary list of active processes for the network of FIG. 2.

Description of Preferred Embodiments

It is to be understood that all program code and data used to implement the inventive methods reside on computer readable media and run on one or more computer systems including standard computer components and operating systems such as UNIX, as known in the art. Furthermore the invention can be implemented on a standalone computer, a client computer communicating with a server computer, or the software components necessary to implement the inventive methods can be distributed among computers on a network such as an intranet or on the Internet. Although the following examples describe the inventive methods implemented in the Java programming language, it is to be understood that the inventive methods can be performed by software written in other programming languages as known in the art, including, but not limited to, languages such as C, C++, or J2EE.

The present invention involves a method for managing successful completion of a network of processes. The network of processes may be represented as a graph, preferably a directed acyclic graph (DAG). In this representation, the nodes of the graph represent the processes, and the edges of the graph represent events associated with the processes. In general, the topology of the final graph will be unknown prior to completion. Processes may be created and deleted dynamically. Processing will successfully complete only when no more processes are to be created (when every process node is a leaf node). In general, an important aspect of the invention is to determine that processing has completed successfully, and to provide a notification message to this effect. Once the notification message is received, results may then be written to persistent storage.

FIG. 1 illustrates an exemplary Java network architecture 100. This network architecture will now be described with respect to a practical application, but it is to be appreciated that many other such applications are possible.

Consider a financial application in which a process *X* is used to effect purchase of shares of stock. Suppose that a market order for 5,000 shares is received Process *X* might be used to purchase the shares on two different stock exchanges where the stock trades. Let us assume that the process *X* spawns a process *A1* to purchase 3,000 shares on a first exchange and a process *A2* to purchase 2,000 shares on a second exchange. Let us further assume that orders will be filled as shares become available. So, at different times t_1 and t_2, processes *B1* and *B2*, might be created to purchase 1,000 and 2,000 shares, respectively. Likewise, at times t_3 and t_4, processes *B3* and *B4* might be created to each purchase 1,000 shares. Perhaps the process *B3* is unable to purchase the full 1,000 shares

as requested; process *B3* might then spawn a process *C1* to purchase the remainder. The resulting graph is shown in FIG. 1.

As can be seen from the above example, processing is based on the result of an initially unknown distributed graph (i.e., a network) of other processes. In addition, it is to be appreciated that one or more of these additional processes may be running in a different thread. Because of the nature of this processing environment, it is evident that determining successful completion of the network can be difficult. A reliable mechanism for determining that the network has finished is needed.

FIG. 2 illustrates an exemplary Q-Manager 250 for managing successful completion of a network of processes. The Q-Manager 250 records the processing state of the network. Once the network has completed processing, the Q-Manager 250 will notify the root node that the network has finished processing. To allow the Q-Manager 250 to work out when processing has completed, a process of "node discovery" is required throughout the processing of events in the network graph. This means that when the network reads (consumes) and then creates (produces) new events to other processes in the network, the Q-Manager 250 is told about these actions. The method also allows the graph of processing nodes to be dynamically changed and will adapt to changes as they happen. No *a priori* knowledge of the graph is ever required by the root node.

In operation, the Q-Manager 250 maintains a list of active processes (nodes), and the list is updated to reflect the current state of the network. That is, as events are produced, the Q-Manager 250 receives messages that the list is to be updated to reflect the creation of the new processes. Conversely, as events are consumed, the Q-Manager 250 receives messages that the list is to be updated to reflect the deletion of certain nodes

in the list. It is to be appreciated that various types of data structures may be used to implement the list of active processes, including an array, a linked-list, a bitmap, a table, etc. For purposes of maintaining the list, it may be desirable to provide a unique identifier for each of the active processes in the list.

Initially, the Q-Manager 250 would receive a message indicating that a network was about to be created. The Q-Manager 250 would then allocate a new list for the network, and write an entry in the list for the root node (e.g., process *X)*. FIG. 3 shows how the list of active processes for the network of FIG. 2 might change over time. At time t_0, the list would only include an entry for the process *X*. Preferably, messages received by the Q-Manager 250 would be sent by the processes themselves at or prior to production of events.

At time t_1, the Q-Manager 250 might receive a message from the process *X* indicating the creation of a process A1. The list at time t_1 would then be updated by adding an entry for process A1. Then, at time t_2, the Q-Manager 250 might receive a message from the process *X* that a process A2 was to be created. The list would be updated by adding an entry for the process A2. Then, at time t_3, the Q-Manager 250 might receive a message from the process *X* that the process *X* was completed. The list would be updated by deleting the entry for the process *X*.

Next, at time t_4, the Q-Manager 250 might receive a message from the process A1 that the process A1 was completed. The list would then be updated by a deleting the entry for process A1. At time t_5, the Q-Manager 250 might receive a message from the process A2 that a process B1 was to be created. The list would then be updated by adding an entry for the process B1. At time t_6, the Q-Manager 250 might receive a message

from the process A2 that a process B3 was to be created. The list would then be updated by adding an entry for the process B3. At time t_7, the Q-Manager 250 might receive a message from the process A2 that a process B2 was to be created. The list would then be updated by adding an entry for the process B2.

Next, at time t_8, the Q-Manager 250 might receive a message from the process A2 that the process A2 was completed. The list would then be updated by deleting the entry for process A2. At time t_9, the Q-Manager 250 might receive a message from the process B1 that the process B1 was completed. The list would then be updated by deleting the entry for process B1. At time t_{10}, the Q-Manager 250 might receive a message from the process B3 that the process B3 was completed. The list would then be updated by deleting the entry for process B3. Finally, at time t_{11}, the Q-Manager 250 might receive a message from the process B2 that the process B2 was completed. The list would then be updated by deleting the entry for the process B2. At this point, the list would be empty.

Once the list becomes empty, the Q-Manager 250 then would then generate a message to signal successful completion of processing. Preferably, the Q-Manager 250 would send the notification message to the root node (process X). At this point, it would be safe to save result information to persistent storage (e.g., to a disk storage device).

Although illustrative embodiments of the present invention have been described herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various other changes and modifications may be affected therein by one skilled in the art without departing from the scope or spirit of the invention.